In this lesson, we walked through various examples of issues you can debug based on error messages, loglines and stack traces.

We have also touched on another very common issue with Spark jobs that can be harder to address: everything working fine but just taking a very long time. So what do you do when your Spark job is (too) slow?

**Insufficient resources**

Often while there are some possible ways of improvement, processing large data sets just takes a lot longer time than smaller ones even without any big problem in the code or job tuning. Using more resources, either by increasing the number of executors or using more powerful machines, might just not be possible. When you have a slow job it’s useful to understand

* how much data you’re actually processing (compressed file formats can be tricky to interpret),
* if you can decrease the amount of data to be processed by filtering or aggregating to lower cardinality,
* and if resource utilization is reasonable.

There are many cases where different stages of a Spark job differ greatly in their resource needs: loading data is typically I/O heavy, some stages might require a lot of memory, others might need a lot of CPU. Understanding these differences might help to optimize the overall performance. Use the Spark UI and logs to collect information on these metrics.

If you run into out of memory errors you might consider increasing the number of partitions. If the memory errors occur over time you can look into why the size of certain objects is increasing too much during the run and if the size can be contained. Also, look for ways of freeing up resources if garbage collection metrics are high.

Certain algorithms (especially ML ones) use the driver to store data the workers share and update during the run. If you see memory issues on the driver check if the algorithm you’re using is pushing too much data there.

**Data skew**

If you drill down on the Spark UI to the task level you can see if certain partitions process significantly more data than others and if they are lagging behind. Such symptoms usually indicate a skewed data set. Consider implementing the techniques mentioned in this lesson:

* add an intermediate data processing step with an alternative key
* adjust the spark.sql.shuffle.partitions parameter if necessary

The problem with data skew is that it’s very specific to a data set. You might know ahead of time that certain customers or accounts are expected to generate a lot more activity but the solution for dealing with the skew might strongly depend on how the data looks like. If you need to implement a more general solution (for example for an automated pipeline) it’s recommended to take a more conservative approach (so assume that your data will be skewed) and then monitor how bad the skew really is.

**Inefficient queries**

Once your Spark application works it’s worth spending some time to analyze the query it runs. You can use the Spark UI to check the DAG and the jobs and stages it’s built of.

Spark’s query optimizer is called Catalyst. While Catalyst is a powerful tool to turn Python code to an optimized query plan that can run on the JVM it has some limitations when optimizing your code. It will for example push filters in a particular stage as early as possible in the plan but won’t move a filter across stages. It’s your job to make sure that if early filtering is possible without compromising the business logic than you perform this filtering where it’s more appropriate.

It also can’t decide for you how much data you’re shuffling across the cluster. Remember from the first lesson how expensive sending data through the network is. As much as possible try to avoid shuffling unnecessary data. In practice, this means that you need to perform joins and grouped aggregations as late as possible.

When it comes to joins there is more than one strategy to choose from. If one of your data frames are small consider using broadcast hash join instead of a hash join.

**Further reading**

Debugging and tuning your Spark application can be a daunting task. There is an ever growing community out there though always sharing new ideas and working on improving Spark itself and tooling that makes using Spark easier. So if you have a complicated issue don’t hesitate to reach out to others (via user mailing lists, forums, and Q&A sites).

You can find more information on tuning [Spark](https://spark.apache.org/docs/latest/tuning.html) and [Spark SQL](https://spark.apache.org/docs/latest/sql-performance-tuning.html) in the documentation